An Application-Based Performance Characterization of the Columbia Supercluster

Rupak Biswas, M. Jahed Djomehri, Robert Hood, Haoqiang Jin, Cetin Kiris, Subhash Saini
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035
{rbiswas,mdjomehri,rhood,hjin,ckiris,ssaini}@mail.arc.nasa.gov

Abstract

Columbia is a 10,240-processor supercluster consisting of 20 Altix nodes with 512 processors each, and currently ranked as the second-fastest computer in the world. In this paper, we present the performance characteristics of Columbia obtained on up to four computing nodes interconnected via the InfiniBand and/or NUMAlink4 communication fabrics. We evaluate floating-point performance, memory bandwidth, message passsing communication speeds, and compilers using a subset of the HPC Challenge benchmarks, and some of the NAS Parallel Benchmarks including the multi-zone versions. We present detailed performance results for three scientific applications of interest to NASA, one from molecular dynamics, and two from computational fluid dynamics. Our results show that both the NUMAlink4 and the InfiniBand hold promise for application scaling to a large number of processors.

Keywords: SGI Altix, multi-level parallelism, HPC Challenge benchmarks, NAS Parallel Benchmarks, molecular dynamics, multi-block overset grids, computational fluid dynamics

1 Introduction

During the summer of 2004, NASA began the installation of Columbia, a 10,240-processor SGI Altix supercomputer at its Ames Research Center. Columbia is a supercluster comprised of 20 nodes, each containing 512 Intel Itanium2 processors and running the Linux operating system. In October of that year, the machine achieved 51.9 Tflop/s on the Linpack benchmark, placing it second on the November 2004 Top500 list [18]. In the ensuing time, we have run a variety of benchmarks and scientific applications on Columbia in an attempt to critically characterize its parallel performance.

In this paper, we present the performance characteristics obtained on up to four computing nodes interconnected via the InfiniBand and/or NUMAlink4 communication fabrics. We first evaluate floating-point performance, memory bandwidth, and message passsing communication speeds using a subset of the HPC Challenge benchmarks [7]. Next, we analyze performance using some of the NAS Parallel Benchmarks [14], particularly the new multi-zone version [9]. Finally, we present detailed performance results for three scientific applications, one from molecular dynamics, and two from state-of-the-art computational fluid dynamics (CFD), both compressible and incompressible multi-block overset grid Navier-Stokes applications [3, 11]. One current problem of significant interest to NASA that involves these applications is the Crew Exploration Vehicle, which will require research and development in several disciplines such as propulsion, aerodynamics, and design of advanced materials.

2 The Columbia Supercluster

Introduced in early 2003, the SGI Altix 3000 systems are an adaptation of the Origin 3000, which use SGI's NUMAflex global shared-memory architecture. Such systems allow access to all data directly and efficiently, without having to move them through I/O or networking bottlenecks. The NUMAflex design enables the processor, memory, I/O, interconnect, graphics, and storage to be packaged into modular components, called "bricks." The primary difference between the Altix and the Origin systems is the C-Brick, used for the processor and memory. This computational building block for the Altix 3700 consists of four Intel Itanium2 processors (in two nodes), 8GB of local memory, and a two-controller ASIC called the Scalable Hub (SHUB). Each C-Brick shares a peak bandwidth of 3.2 GB/s via the NUMAlink interconnection. Each SHUB interfaces to two CPUs in one node, along with memory, I/O devices, and other SHUBs. The Altix cache-coherency protocol is implemented in the SHUB that integrates both the snooping operations of the Itanium2 and the directory-based scheme used across the NUMAlink interconnection fabric. A load/store cache miss causes the data to be communicated via the SHUB at a cache-line granularity and automatically replicated in the local cache.

The predominant CPU on Columbia is an implementation of the 64-bit Itanium2 architecture, operates at 1.5 GHz, and is capable of issuing two multiply-adds per cycle for a peak performance of 6.0 Gflop/s. The memory hierarchy consists of 128 floating-point registers and three on-chip data caches (32KB L1, 256KB L2, and 6MB L3). The Itanium2 cannot store floating-point data in L1, making register loads and spills a potential source of bottlenecks; however, a relatively large register set helps mitigate this issue. The superscalar processor implements the Explicitly Parallel Instruction set Computing (EPIC) technology where instructions are organized into 128-bit VLIW bundles. The Altix 3700 platform uses the NUMAlink3 interconnect, a high-performance custom network with a fat-tree topology that enables the bisection bandwidth to scale linearly with the number of processors.

Columbia is configured as a cluster of 20 SGI Altix nodes (or boxes), each with 512 processors and approximately 1TB of global shared-access memory. Of these 20 nodes, 12 are model 3700 and the remaining eight are model 3700BX2. The BX2 node is essentially a double-density version of the 3700. Each BX2 C-Brick thus contains eight processors, 16GB local memory, and four SHUBs, doubling the processor count in a rack from 32 to 64 and thereby packing more computational power in the same space. The BX2 C-Bricks are interconnected via NUMAlink4, yielding a peak bandwidth of 6.4 GB/s that is twice the bandwidth between bricks on a 3700. In addition, five of the Columbia BX2's use 1.6 GHz (rather than 1.5 GHz) parts and 9MB L3 caches. Table 1 summarizes the main characteristics of the 3700 and BX2 nodes used in Columbia.

Characteristics	3700	BX2
Architecture	NUMAflex, SSI	NUMAflex, SSI
# Processors	512	512
Packaging	32 CPUs/rack	64 CPUs/rack
Processor	Itanium2	Itanium2
Clock/L3 cache	1.5 GHz/6 MB	1.5 GHz/6 MB (a) 1.6 GHz/9 MB (b)
Interconnect	NUMAlink3	NUMAlink4
Bandwidth	3.2 GB/s	6.4 GB/s
Memory	1 TB	1 TB
Th. peak perf.	3.07 Tflop/s	3.07 Tflop/s (a) 3.28 Tflops (b)

Table 1. Characteristics of the two types of Altix nodes used in Columbia.

Two communication fabrics connect the 20 Altix systems: an InfiniBand switch [19] provides low-latency MPI communication, and a 10-gigabit Ethernet switch provides user access and I/O communications. InfiniBand is a revolutionary, state-of-the-art technology that defines very high-speed networks for interconnecting compute and I/O nodes [8]. It is an open industry standard for designing high-performance compute clusters of PCs and SMPs. Its high peak bandwidth and comparable minimum latency distinguish it from other competing network technologies such as Quadrics and Myrinet [12]. Four of the 1.6 GHz BX2 nodes are linked with NUMAlink4 technology to allow the global shared-memory constructs to significantly reduce interprocessor communication latency. This 2,048-processor subsystem within Columbia provides a 13 Tflop/s peak capability platform.

A number of programming paradigms are supported on Columbia, including the standard OpenMP and MPI, SGI SHMEM, and Multi-Level Parallelsim (MLP). MPI and SHMEM are provided by SGI's Message Passing Toolkit (MPT), while C/C++ and Fortran compilers from Intel support OpenMP. The MLP library was developed by Taft at NASA Ames [17]. Both OpenMP and MLP can take advantage of the globally shared memory within an Altix node. Both MPI and SHMEM can be used to communicate between Altix nodes connected with the NUMAlink interconnect; however, communication over the InfiniBand switch requires the use of MPI. Because of the limitation on the number of InfiniBand connections through InfiniBand cards installed on each node, the number of per-node MPI processes is confined by

$$k \leq \sqrt{\frac{N_{\text{cards}} \times N_{\text{connections}}}{n-1}}$$

where n (≥ 2) is the number of Altix nodes involved. Currently on Columbia, $N_{\text{cards}} = 8$ per node and $N_{\text{connections}} = 64$K per card. Thus, a pure MPI code can only fully utilize up to three Altix nodes. A hybrid (e.g. MPI+OpenMP) version of applications would be required for runs using four or more nodes.

3 Benchmarks and Applications

We utilize a spectrum of microbenchmarks, synthetic benchmarks, and scientific applications in order to critically characterize Columbia performance. These are briefly described in the following subsections.

3.1 HPC Challenge Microbenchmarks

We elected to test basic system performance characteristics such as floating-point operations, memory bandwidth, and message passing communication speeds using a subset of the HPC Challenge (HPCC) benchmark suite [7]. In particular, we used the following components:

- We tested optimum floating-point performance with DGEMM, a double-precision matrix-matrix multiplication routine that uses a level-3 BLAS package on the Altix. The input arrays are sized so as to use about 75% of the memory available on the subset of the CPUs being tested.

- The STREAM benchmark component tests memory bandwidth by doing simple operations on very long vectors. There are four vector operations measured: copy, scale by multiplicative constant, add, and triad (multiply by scalar and add). As with the DGEMM benchmark, the vectors manipulated are sized to use about 75% of the memory available.

- We evaluated message passing performance in a variety of communication patterns with the HPCC version of b_eff [6]. The test measures latency and bandwidth using ping-pong and two rings: one using a "natural" ordering where communication takes place between processes with adjacent ranks in MPI_COMM_WORLD, and one using a random ordering. For ping-pong, we use the "average" results reported by the benchmark; for the rings, the benchmark reports a geometric mean of the results from a number of trials.

While these benchmarks will likely not be completely indicative of application performance, they can be used to help explain application timing anomalies when they occur.

3.2 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are well-known problems for testing the capabilities of parallel computers and parallelization tools. They were derived from computational fluid dynamics (CFD) codes and are widely recognized as a standard indicator of parallel computer performance. The original NPB suite consists of five kernels and three simulated CFD applications, given as a "pencil-and-paper" specifications in [1]. The five kernels mimic the computational core of five numerical methods, while the three simulated applications reproduce much of the data movement and computation found in full CFD codes. Reference implementations were subsequently provided as NPB2 [2], using MPI as the parallel programming paradigm, and later expanded to other programming paradigms (such as OpenMP).

Recent effort in NPB development was focused on new benchmarks, including the new multi-zone version, called NPB-MZ [9]. While the original NPB exploits fine-grain parallelism in a single zone, the multi-zone benchmarks stress the need to exploit multiple levels of parallelism for efficiency and to balance the computational load.

For evaluating the Columbia system, we selected a subset of the benchmarks: three kernels (MG, CG, and FT), one simulated application (BT), and two multi-zone benchmarks (BT-MZ and SP-MZ) [2, 9]. These cover five types of numerical methods found in many scientific applications. Briefly, MG (multi-grid) tests long- and short- distance communication, CG (conjugate gradient) tests irregular memory access and communication, FT (fast Fourier transform) tests all-to-all communication, BT (block-triadiagonal solver) tests nearest neighbor communication, and BT-MZ (uneven sized zones) and SP-MZ (even sized zones) test both coarse- and fine-grain parallelism and load balance. For our experiments, we use both MPI and OpenMP implementations of the four original NPBs and the hybrid MPI+OpenMP implementation of the NPB-MZ from the latest NPB3.1 distribution [14]. To stress the processors, memory, and network of the Columbia system, we introduced two new classes of problem sizes for the multi-zone benchmarks: Class E (4096 zones, $4224 \times 3456 \times 92$ aggregated grid size) and Class F (16384 zones, $12032 \times 8960 \times 250$ aggregated grid size).

3.3 Molecular Dynamics

Molecular dynamics [15] is a powerful technique for studying the structure of solids, liquids and gases. It involves calculating the forces acting on the atoms in a molecular system using Newton's equations of motion and studying their trajectories as a function of time. After integrating for some time when sufficient information on the motion of the individual atoms has been collected, one uses statistical methods to deduce the bulk properties of the material. These properties may include the structure, thermodynamics, and transport properties. In addition, molecular dynamics can be used to study the detailed atomistic mechanisms underlying these properties and compare them with theory. It is a valuable computational tool to bridge between experiment and theory.

In our Columbia performance study we use a generic molecular dynamics code based on the Velocity Verlet algorithm, a sophisticated integrator designed to further improve the velocity evaluations. However, it is computationally more expensive than other integration algorithms like Verlet or leap-frog schemes. The Velocity Verlet algorithm provides both the atomic positions and velocities at the same instant of time, and therefore is regarded as the most complete form of the Verlet algorithm.

To parallelize the algorithm, we use a spatial decomposition method, in which the physical domain is subdivided into small three-dimensional boxes, one for

each processor. At each step, the processors compute the forces and update the positions and velocities of all the atoms within their respective boxes. In this method, a processor needs to know the locations of atoms only in nearby boxes; thus, communication is entirely local. Each processor uses two data structures: one for the atoms in its spatial domain and the other for atoms in neighboring boxes. The first data structure stores atomic positions and velocities, and neighbor linked lists to permit easy deletions and insertions as atoms move between boxes. The second data structure stores only position coordinates of atoms in neighboring boxes. The potential energy between two atoms is modeled by the Lennard-Jones potential. The simulation starts with atoms on a force cubic center (fcc) lattice with randomized velocities at a given temperature. We used a cutoff radius of 5.0 beyond which interactions between atoms are not calculated.

3.4 INS3D: Turbopump Flow Simulations

Computations for unsteady flow through a full scale low-pressure rocket pump are performed utilizing the INS3D computer code [10]. Liquid rocket turbopumps operate under severe conditions and at very high rotational speeds. The low-pressure-fuel turbopump creates transient flow features such as reverse flows, tip clearance effects, secondary flows, vortex shedding, junction flows, and cavitation effects. Flow unsteadiness originated from the inducer is considered to be one of the major contributors to the high frequency cyclic loading that results in cycle fatigue. The reverse flow originated at the tip of an inducer blade travels upstream and interacts with the bellows cavity. To resolve the complex geometry in relative motion, an overset grid approach is employed where the problem domain is decomposed into a number of simple grid components [3]. Connectivity between neighboring grids is established by interpolation at the grid outer boundaries. Addition of new components to the system and simulation of arbitrary relative motion between multiple bodies are achieved by establishing new connectivity without disturbing the existing grids.

The computational grid used for the experiments reported in this paper consisted of 66 million grid points and 267 blocks (or zones). Details of the grid system are shown in Fig. 1. Fig. 2 displays particle traces colored by axial velocity entering the low-pressure fuel pump. The blue particles represent regions of positive axial velocity, while the red particles indicate four back flow regions. The gray particles identify the stagnation regions in the flow.

The INS3D code solves the incompressible Navier-Stokes equations for both steady-state and unsteady

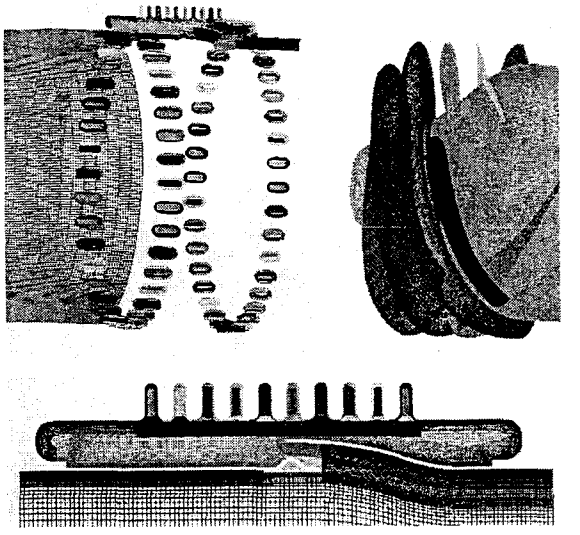

Figure 1. Surface grids for the low pressure fuel pump inducer and the flowliner.

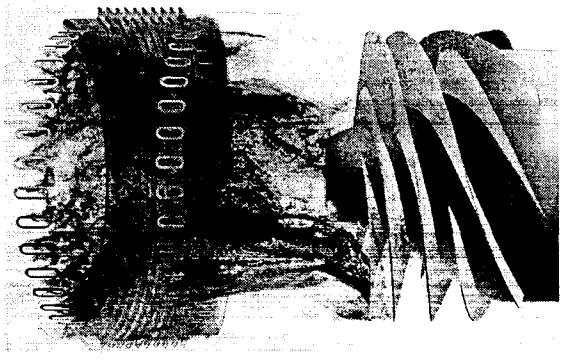

Figure 2. Instantaneous snapshot of particle traces colored by axial velocity values.

flows. The numerical solution requires special attention in order to satisfy the divergence-free constraint on the velocity field. The incompressible formulation does not explicitly yield the pressure field from an equation of state or the continuity equation. One way to avoid the difficulty of the elliptic nature of the equations is to use an artificial compressibility method that introduces a time-derivative of the pressure term into the continuity equation. This transforms the elliptic-parabolic partial differential equations into the hyperbolic-parabolic type. To obtain time-accurate solutions, the equations are iterated to convergence in pseudo-time for each physical time step until the divergence of the velocity field has been reduced below a specified tolerance value. The total number of sub-iterations required varies depending on the problem, time step size, and the artificial compressibility parameter. Typically, the number ranges

from 10 to 30 sub-iterations. The matrix equation is solved iteratively by using a non-factored Gauss-Seidel type line-relaxation scheme, which maintains stability and allows a large pseudo-time step to be taken. More detailed information about the application can be found in [10, 11].

Performance results reported in this paper were obtained for computations carried out using the Multi-Level Parallelism (MLP) paradigm for shared-memory systems [17]. All data communications at the coarsest and finest levels are accomplished via direct memory referencing instructions. The coarsest level parallelism is supplied by spawning off independent processes via the standard UNIX fork. A library of routines is used to initiate forks, to establish shared memory arenas, and to provide synchronization primitives. The boundary data for the overset grid system is archived in the shared memory arena by each process. Fine grain parallelism is obtained by using OpenMP compiler directives.

3.5 OVERFLOW-D: Rotor Vortex Simulations

For solving the compressible Navier-Stokes equations, we selected the NASA production code called OVERFLOW-D [13]. The code uses the same overset grid methodology [3] as INS3D to perform high-fidelity viscous simulations around realistic aerospace configurations. OVERFLOW-D is popular within the aerodynamics community due to its ability to handle complex designs with multiple geometric components. It is explicitly designed to simplify the modeling of problems when components are in relative motion. The main computational logic at the top level of the sequential code consists of a time-loop and a nested grid-loop. Within the grid-loop, solutions to the flow equations are obtained on the individual grids with imposed boundary conditions. Overlapping boundary points or inter-grid data are updated from the previous time step using a an overset grid interpolation procedure. Upon completion of the grid-loop, the solution is automatically advanced to the next time step by the time-loop. The code uses finite difference schemes in space, with a variety of implicit/explicit time stepping.

The hybrid MPI+OpenMP version of OVERFLOW-D takes advantage of the overset grid system, which offers a natural coarse-grain parallelism [4]. A bin-packing algorithm clusters individual grids into groups, each of which is then assigned to an MPI process. The grouping strategy uses a connectivity test that inspects for an overlap between a pair of grids before assigning them to the same group, regardless of the size of the boundary data or their connectivity to other grids. The grid-loop in the parallel implementation is subdivided into two procedures: a group-loop over groups, and a grid-loop over the grids within each group. Since each MPI process is assigned to only one group, the group-loop is executed in parallel, with each group performing its own sequential grid-loop. The inter-grid boundary updates within each group are performed as in the serial case. Inter-group boundary exchanges are achieved via MPI asynchronous communication calls. The OpenMP parallelism is achieved by the explicit compiler directives inserted at the loop level. The logic is the same as in the pure MPI case, only the computationally intensive portion of the code (i.e. the grid-loop) is multi-threaded via OpenMP.

OVERFLOW-D was originally designed to exploit vector machines. Because Columbia is a cache-based superscalar architecture, modifications were necessary to improve performance. The linear solver of the application, called LU-SGS, was reimplemented using a pipeline algorithm [4] to enhance efficiency which is dictated by the type of data dependencies inherent in the solution algorithm.

Our experiments reported here involve a Navier-Stokes simulation of vortex dynamics in the complex wake flow region around hovering rotors. The grid system consisted of 1679 blocks of various sizes, and approximately 75 million grid points. Fig. 3 shows a sectional view of the test application's overset grid system (slice through the off-body wake grids surrounding the hub and rotors) while Fig. 4 shows a cut plane through the computed wake system including vortex sheets as well as a number of individual tip vortices. A complete description of the underlying physics and the numerical simulations pertinent to this test problem can be found in [16].

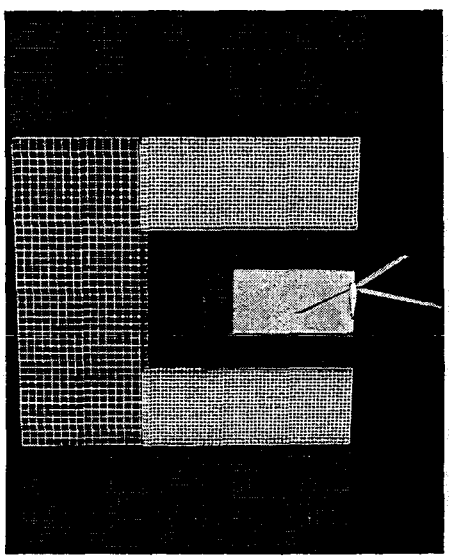

Figure 3. A sectional view of the overset grid system.

Figure 4. Computed vorticity magnitude contours on a cutting plane located 45° behind the rotor blade.

4 Performance Results

We conducted several experiments using microbenchmarks, synthetic benchmarks, and full-scale applications to obtain a detailed performance characterization of Columbia. Results of these experiments are presented in the following subsections.

4.1 3700 vs. BX2

In comparing the performance of the 3700 with two types of BX2, we are assessing the impact of both improved processor speed (coupled with larger L3 cache) and processor interconnect. As a shorthand notation, we will call the BX2 with 1.5 GHz CPUs and 6MB caches a "BX2a". The BX2 with faster clock and larger cache is denoted "BX2b".

4.1.1 HPC Challenge Microbenchmarks

The performance of the DGEMM benchmark showed a correlation with processor speed and cache size rather than processor interconnect. When run on a BX2b, performance (5.75 GFlop/s) improved by 6% versus runs on 3700 or BX2a, which were essentially identical.

The STREAM Triad benchmark showed only 1% better performance on a 3700 versus either type of BX2. Nothing about published architecture differences indicates why this might be the case. For the final version of the paper we will run additional experiments to try and pin down a reason.

The MPI latency and bandwidth results are shown in Fig. 5. For Ping-Pong and Natural Ring, the latencies are remarkably consistent between 3700 and both models of BX2. The Random Ring latency test shows that as average communication distances become further apart (as processor counts increase), the interconnect network improvements in the BX2 take effect.

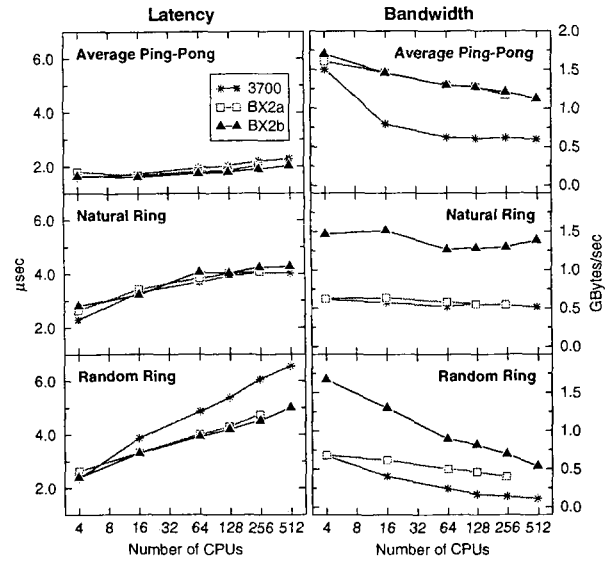

Figure 5. Bandwidth and latency tests on three types of the Columbia nodes.

Bandwidth was correlated either to processor speed or interconnect, depending on the locality of the communication tested. On the Ping-Pong test, where there is some distance between communicating pairs of processes, the interconnect used plays a key role in the bandwidth. In the case of the Natural Ring, where local communication predominates, processor speed is the determining factor. In the Random Ring, where the communications are mostly remote, both processor speed and interconnect show effects for bandwidth.

4.1.2 NAS Parallel Benchmarks

Fig. 6 shows the per-processor Gflop/s rates reported from runs of both MPI and OpenMP versions of CG, FT, MG, and BT benchmarks on three types of the Columbia nodes. As was seen from the HPCC microbenchmarks in the previous section, the double density packing for BX2 produces shorter latency and higher bandwidth in NUMAlink access. The effect of higher bandwidth of BX2 on OpenMP performance is evident: the four OpenMP benchmarks scaled much better on both types of BX2 than on 3700 when the number of threads is four or more. With 128 threads, the difference can be as large as 2x for both FT and BT. The clock speed and cache size had less effect on the OpenMP codes.

Bandwidth effect on MPI performance is less profound until a larger number of processes ($\geq$32). Observe that on 256 processors, FT runs about twice as fast on BX2 than on 3700, indicating the importance of bandwidth for the all-to-all communication used in the benchmark. At about 64 processors, both MG and BT

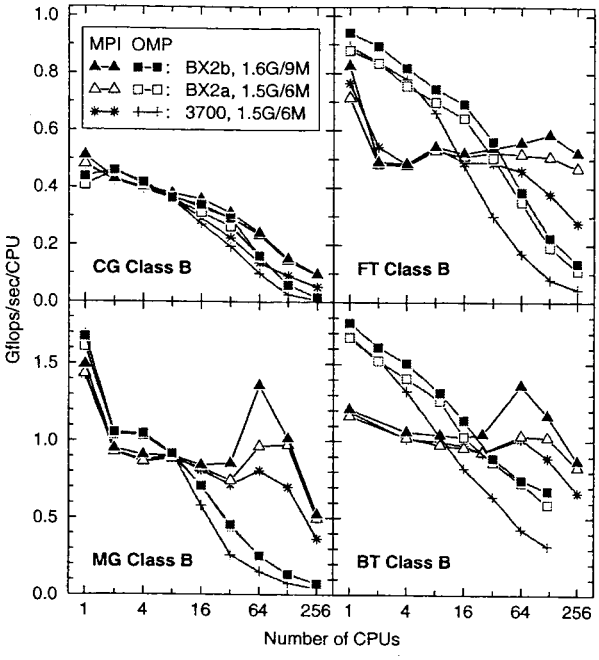

Figure 6. NPB performance comparison on three types of the Columbia nodes.

exhibit a performance jump (~50%) on BX2b comparing to BX2a. We believe this is a result of a larger L3 cache on the BX2b node.

Overall, OpenMP versions of NPB demonstrated better performance on a small number of CPUs, but MPI versions scaled much better. The OpenMP codes are less sensitive to cache size, but more to the communication bandwidth. The impact of processor speed on performance is generally small.

4.1.3 INS3D

Computations to test the scalability of the INS3D code on Columbia were performed using the 3700 and BX2b processors along with the 7.1 and 8.1 F90 compilers. Initial computations using one MLP group and one OpenMP thread with the various processor and compiler options were used to establish the baseline runtime for one physical time step of the solver, where 720 such time steps are required to complete one inducer rotation. Next, a fixed number of 36 MLP groups was chosen along with various numbers of OpenMP threads (1, 2, 4, 8, 12, or 14). The average runtime per iteration is shown in Table 2. Observe that the BX2b demonstrates approximately 50% faster iteration time. Note the scalability for fixed MLP groups and varying OpenMP threads is good, but begins to decay as the number of threads increases beyond eight. Further scaling can be accomplished by fixing the number of threads and varying the number of MLP groups until the load balancing begins to fail. Unlike varying the OpenMP threads which does not affect the convergence rate of INS3D, varying the number of MLP groups may deteriorate convergence. This will lead to more iterations even though faster runtime per iteration is achieved.

INS3D		
	3700	BX2b
P	Exec (sec)	Exec (sec)
1	39230.0	26430.0
36 (36×1)	1223.0	825.2
72 (36×2)	796.0	508.4
144 (36×4)	554.2	331.8
288 (36×8)	454.7	287.7
432 (36×12)	409.1	—
504 (36×14)	—	247.6

Table 2. INS3D performance on 3700 and BX2b.

4.1.4 OVERFLOW-D

The performance of OVERFLOW-D was also evaluated on Columbia using the 3700 and BX2b processors. Table 3 shows communication and total execution times of the application per time step when using the 8.1 Fortran compiler. Note that a typical production run requires on the order of 50,000 such time steps. For various number of processors we report the time from the best combination of processes and threads.

OVERFLOW-D				
	3700		BX2b	
P	Comm (sec)	Exec (sec)	Comm (sec)	Exec (sec)
1	0.22	151.2	0.21	126.4
4 (4×1)	1.2	38.4	0.82	32.0
16 (16×1)	2.4	16.2	0.41	9.0
32 (32×1)	1.9	7.8	0.42	4.6
64 (64×1)	1.6	5.5	0.45	2.5
128 (128×1)	1.0	4.4	0.36	1.6
256 (128×2)	1.0	3.1	0.42	1.3
508 (254×2)	1.9	3.8	0.70	1.1

Table 3. OVERFLOW-D performance on 3700 and BX2b.

Observe from Table 3 that execution time on BX2b is significantly smaller compared to 3700 (e.g. more than a factor of 3x on 508 CPUs). On average, OVERFLOW-D runs almost 2x faster on the BX2b than the 3700. In addition, the communication time is also reduced by more than 50%.

The performance scalability on the 3700 is reason-

ably good up to 64 processors, but flattens beyond 256. This is due to the small ratio of grid blocks to the number of MPI tasks that makes balancing computational workload extremely challenging. With 508 MPI processes and only 1679 blocks, it is difficult for any grouping strategy to achieve a proper load balance. Various load balancing strategies for overset grids are extensively discussed in [5].

Another reason for poor 3700 scalability on large processor counts is insufficient computational work per processor. This can be verified by examining the ratio of communication to execution time in Table 3. This ratio is about 0.3 for 256 processors, but increases to more than 0.5 on 508 CPUs. For our test problem consisting of 75 million grid points, there are only about 150 thousand grid points per MPI task, which is too little for Columbia's fast processors compared to the communication overhead. The test problem used here was initially built for production runs on platforms having fewer processors with smaller caches and slower clock rates. We plan to run a much larger grid system for the final paper.

Scalability on the BX2b is significantly better. For example, OVERFLOW-D efficiency for 128, 256, and 508 processors is 61%, 37%, and 27% (compared to 26%, 19%, and 7% on the 3700). In spite of the same load imbalance problem, the enhanced bandwidth on the BX2b significantly reduces the communication times. The increased bandwidth is particularly important at the coarse-grain level of OVERFLOW-D, which has an all-to-all communication pattern every time step. This is consistent with our experiments conducted on the NPBs and reported in Sec 4.1.2. The reduction in the BX2b computation time can be attributed to its larger L3 cache.

4.2 CPU "Stride"

When performing STREAM benchmarking on 15 of the 20 nodes of Columbia in October 2004, we observed, not unexpectedly, that the results scaled linearly from 2 to 7500 CPUS—achieving ~2 GB/s per CPU. However, when run on a single processor, the benchmark registered ~3.8 GB/s. We hypothesized that this is due to each memory bus being shared by two processors. To verify that and to understand what other behavior might be due to that (or other resource) sharing, we ran the HPCC benchmarks in a "spread out" or strided fashion, using every second or every fourth CPU.

The DGEMM benchmark demonstrated differences of less than 0.5%—showing that this benchmark is not substantially affected by sharing the memory bus. As expected, at a CPU stride of either 2 or 4, the STREAM benchmark produced per-processor numbers equivalent to the 1-CPU case. In the case of Triad, the bandwidth is 1.9x higher than when processes are assigned to CPUs in a dense fashion. The latency-bandwidth results were less dramatic. The numbers for Ping-Pong and Random Ring were slightly worse for spread-out CPUs. The results for Natural Ring were less conclusive. There was a small improvement in latency but none for bandwidth.

4.3 Pinning

Application performance on NUMA architectures like an Altix node depends on data and thread placement onto CPUs. Improper initial data placement or unwanted migration of threads between processors can increase memory access time, thus degrading performance. The performance impact of using thread-to-processor pinning on applications, in particular hybrid codes, can sometimes be substantial. This is illustrated by the results shown in Fig. 7 for the hybrid MPI+OpenMP SP-MZ code running with and without pinning. Each curve is associated with runs for a given total number of CPUs, but varying the number of OpenMP threads per MPI process. Observe that pinning improves performance substantially in the hybrid mode when processes spawn multiple threads. The impact becomes even more profound as the number of CPUs increases. Pure process mode (e.g. 64×1) is less influenced by pinning.

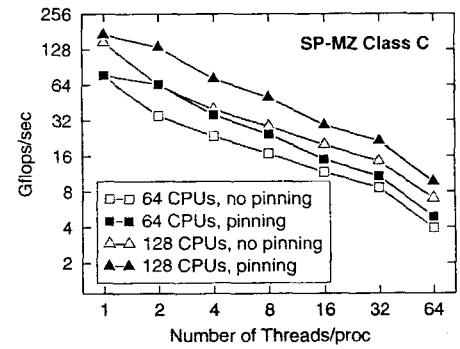

Figure 7. Pinning versus no pinning for SP-MZ Class C running on BX2b.

A user has at least three different methods for pinning on the Altix:

1. Set environment variables (`MPI_DSM_DISTRIBUTE` and `MPI_DSM_CPULIST`) for MPI codes,
2. Use the data placement tool, `dplace`, for either MPI or OpenMP codes, and/or
3. Insert system calls in the user's code, in particular, for hybrid implementations.

All other results reported in this paper have pinning applied, either using method 2 or a combination of methods 2 and 3.

4.4 Compiler Versions

There are at least four different versions of the Intel compilers installed on the Columbia system: 7.1(.042), 8.0(.070), 8.1(.026), and 9.0(.012)beta. Although 8.1 is the latest official release, the default compiler is still set to 7.1 for various reasons. A user can apply the `module` command to select a particular version of the compiler. For evaluation purposes, a beta version of the 9.0 compiler is also included.

The performance impact of different compiler versions was examined with the four OpenMP NPB benchmarks and the results are shown in Fig. 8. All tests were conducted on a BX2b node with the `-O3 -openmp` compilation flag. We noted that the compiler performance seems to be application dependent, although the 8.0 version produced the worst results in most cases. All the compilers gave similar results on the CG benchmark. The beta version of 9.0 performed very well on FT. For MG, between 32 and 128 threads (or CPUs) the 8.1 and 9.0b compilers outperformed the 7.1 and 8.0; however, below 32 threads, the 7.1 and 8.0 compilers performed 20–30% better than the other two. The scaling also turns around above 128 threads.

Overall, the 7.1 compiler produced consistently better performance for most the benchmarks, in particular for a small number of threads. As a result, 7.1 was used for the remaining NPB tests in this report.

Using the BX2b processor, the INS3D flow solver was compiled and run using both the 7.1 and 8.1 versions of the Fortran compiler with negligible difference in runtime per iteration (see Table 4). Evaluations for OVERFLOW-D where only performed on the 3700 node. Timing results with 7.1 are superior to those with 8.1 by 20–40% when running on less than 64 processors, but almost identical on larger counts.

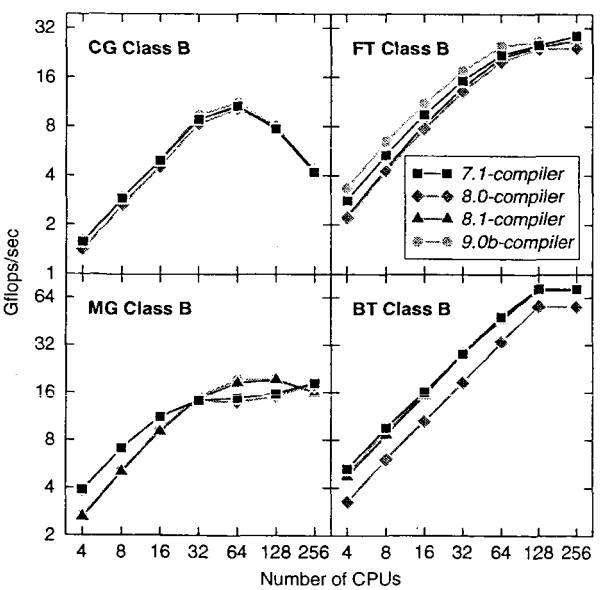

Figure 8. Performance comparison of four compiler versions.

INS3D			OVERFLOW-D		
P	7.1 (sec)	8.1 (sec)	P	7.1 (sec)	8.1 (sec)
1	26430.0	25637.1	1	111.3	151.2
			4	28.4	38.4
			16	11.2	16.2
36	825.2	783.8	32	6.5	7.8
72	508.4	487.7	64	5.1	5.5
144	331.8	324.4	128	4.5	4.4
288	287.7	270.4	256	3.1	3.1
504	247.6	244.9	508	3.7	3.8

Table 4. INS3D and OVERFLOW-D performance using Intel Fortran compilers 7.1 and 8.1

4.5 Processes and Threads

We examined the performance of hybrid codes under various MPI process and OpenMP thread combinations within one Altix node. The results for the BT-MZ benchmark are shown in Fig. 9. For a given number of OpenMP threads (left panel in Fig. 9), MPI scales very well, almost linearly up to the point where load imbalancing becomes a problem. On the other hand, for a given number of MPI processes (right panel of Fig. 9), OpenMP scaling is very limited: except for two threads, OpenMP performance drops quickly as the number of threads increases.

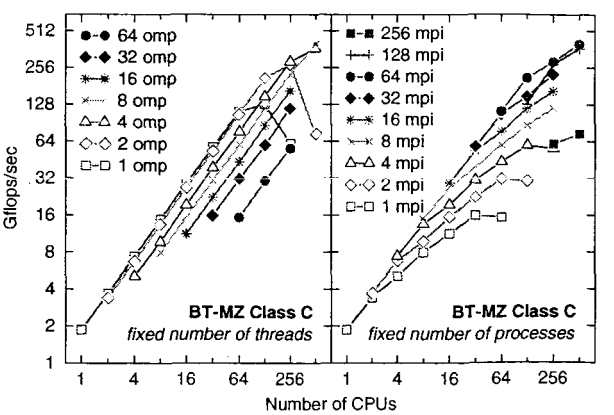

Figure 9. Effects of varying processes and threads on the BT-MZ benchmark.

4.6 Multinode Execution

We next reran a subset of our experiments on up to four BX2b Altix nodes. These results are presented in the following subsections.

4.6.1 HPC Challenge Microbenchmarks

The internode communication network played a very minor role (less than 0.5%) in the performance of DGEMM, and none at all in STREAM. The latter result is not surprising since the benchmark does not measure communication time.

In the tests of MPI latency and bandwidth (see Fig. 10) it is clear that NUMAlink4 performs much better than InfiniBand between nodes. The latency results show a substantial penalty for InfiniBand across two nodes and even worse performance across four nodes. In the case of Ping-Pong, this can probably be explained by the increase in the number of "off-node" pairs that get tested when four nodes are used. The Natural Ring latency results show a smaller penalty for the increase from two to four nodes. This decreased penalty is understandable because the benchmark reports the *worst-case* process-to-process latency for the entire ring communication, while we use an *average* point-to-point latency in Ping-Pong.

The bandwidth results for Ping-Pong and Natural Ring show a similar correlation to out-of-node communications. In the case of Ping-Pong, since we are reporting the average of a series of point-to-point bandwidth experiments, there is a falloff in InfiniBand performance from two to four because the likelihood of a non-local pairing increases. For Natural Ring, the two- and four-node tests yielded similar results.

The latency and bandwidth results from the Random Ring tests show severe problems with scalability of InfiniBand. It is possible that our results were affected by a configuration problem, so we will repeat them (and vary several configuration parameters) for the final version of the paper.

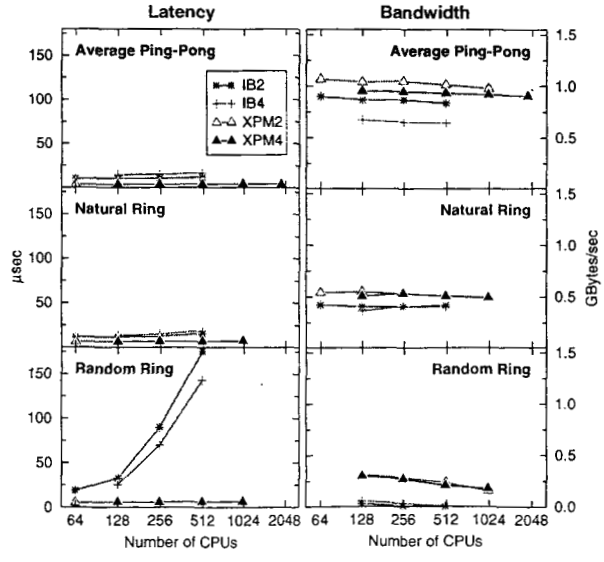

Figure 10. Bandwidth and latency tests on three types of the Columbia nodes.

4.6.2 NAS Parallel Benchmarks

The hybrid MPI+OpenMP codes of BT-MZ and SP-MZ were also tested across four Columbia nodes connected with both the NUMAlink4 network and the InfiniBand switch. We used the Class E problem (4096 zones, 1.3 billion aggregated grid points) for these tests. The top row of Fig. 11 compares the per-CPU Gflop/s rates obtained from runs using NUMAlink4 with those from within a single Altix BX2b node. The two sets of data represent runs with one and two OpenMP threads per MPI process, respectively. For 512 CPUs or less, the NUMAlink4 results are comparable to or even better than the in-node results. In particular, the performance of 512-processor runs in a single node dropped by 10–15%, primarily because these runs also used the CPUs that were allocated for systems software (called *boot cpuset*), which interfered with our tests. Reducing the number of CPUs to 508 improves the BT-MZ performance.

Since MPI is used for coarse-grain parallelism among zones for the hybrid implementations, load balancing for SP-MZ is trivial as long as the number of zones is divisible by the number of MPI processes. The uneven-size

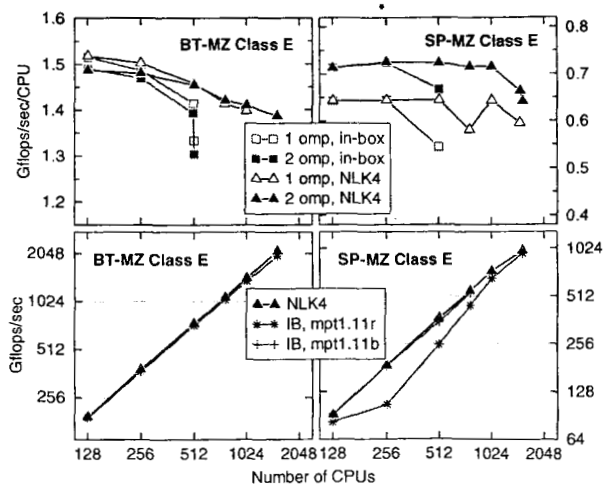

Figure 11. Comparison of NPB-MZ performance under three different networks.

zones in BT-MZ allows more flexible choice of the number of MPI processes; however, as the number of CPUs increases, OpenMP threads may be required to get better load balance (and therefore better performance). This is evident from the BT-MZ results in Fig. 11. There is about 11% performance improvement from runs using two OpenMP threads versus one (e.g. 256×2 vs. 512×1) for the SP-MZ benchmark. This effect could be attributed to less MPI communication when two threads are used. The performance drop for SP-MZ at 768 and 1536 processors can be explained by load imbalance for these CPU counts.

The bottom row of Fig. 11 compares the total Gflop/s rates from runs using NUMAlink4 with those from using the InfiniBand, taking the best process-thread combinations. Observe a close-to-linear speedup for BT-MZ. The InfiniBand results are only about 7% worse. On the other hand, we noticed anomaly in InfiniBand performance for SP-MZ when a released SGI MPT runtime library (mpt1.11r) was used. In fact, on 256 processors, the InfiniBand result is 40% slower than NUMAlink4, but the InfiniBand performance improves as the number of CPUs increases. We used a beta version of the MPT library (mpt1.11b) and reran some of the data points. These results are also included in Fig. 11 for SP-MZ. As we can see, the beta version of the library produced InfiniBand results that are very close in performance to the NUMAlink4 results. We are actively working with SGI engineers to find the true cause of the anomaly.

4.6.3 Molecular Dynamics

Table 5 shows the wall clock time as a function of the number of processors for the molecular dynamics code. This is a weak scaling exercise: we assign 64,000 atoms to each processor, and thus scale the problem size with the processor count. For 2040 processors, we simulated 130.56 million atoms. The entire simulation was run for 100 steps. Results show almost perfect scalability all the way up to 2040 processors. The communication costs are insignificant for this test case; however, they could increase if the simulation were run for long durations and workloads became unbalanced.

Wall clock time per step (sec)		
P	# particles	time
64	4,096,000	21.54
128	8,192,000	21.55
256	16,384,000	21.58
512	32,768,000	21.82
1024	65,536,000	21.96
1536	98,304,000	21.70
2040	130,560,000	21.61

Table 5. Performance of molecular dynamics code using NumaLink4 interconnection.

4.6.4 OVERFLOW-D

Table 6 presents performance experiments conducted on multiple BX2b nodes. The column denoted as "# of Nodes" refers to thee number of BX2b nodes used. The communication and execution times are reported for the same runs via both NUMAlink4 and InfiniBand interconnects, using the Intel Fortran compiler 8.1.

P	# of Nodes	Cross nodes; BX2b			
		NUMAlink4		InfiniBand	
		comm (sec)	exec (sec)	comm (sec)	exec (sec)
256 (256×1)	2	0.38	1.5	0.35	1.5
256 (256×1)	4	0.38	1.5	0.36	1.8
508 (508×1)	2	0.39	2.0	0.35	2.3
508 (508×1)	4	0.14	1.4	—	—
1016 (508×2)	4	0.14	1.4	0.17	1.9
1464 (366×4)	3	0.19	2.9	0.15	2.8
1464 (732×2)	3	0.25	2.6	—	—
1972 (493×4)	4	0.17	2.1	0.13	2.1
2032 (508×4)	4	0.16	2.1	0.12	2.3

Table 6. Performance of OVERFLOW-D across multiple BX2b nodes via NumaLink and InfiniBand interconnection.

The total execution times obtained via NUMAlink4 are generally about 10% better; however, the reverse appears to be true for the communication times. We did not find any pronounced increase in the execution and communication timing for the same total number of processors when distributed across multiple nodes. This suggests that performance scalability over many nodes is not affected by the type of the interconnect for this application.

OVERFLOW-D has significant I/O requirements at runtime. Due to the lack of a shared file system among the Columbia nodes at this time, a less efficient file system was used. Some of the performance results may therefore have been affected to some extent by I/O activities. We expect to have this problem resolved in the final paper. Also, an overset grid system suitable in size and the number of blocks to fully exploit the computational capability of Columbia is under construction. We will use this much larger system for the final paper.

5 Summary and Conclusions

Our benchmarking on the Columbia supercluster demonstrated several features about single-box SGI Al-

tix performance. First, the presence of NUMAlink4 on the BX2 nodes provides a large performance boost for MPI and OpenMP applications. Furthermore, when the processor speed and cache size are enhanced (as is the case on those nodes we call BX2b's), there is another significant improvement in performance. As was the case on the SGI Origins, process and thread pinning continues to be critical to performance. Among the four versions of the Intel compiler that we tested, there is no clear winner—performance seems to vary with application.

When multiple Altix nodes are combined into a capability cluster, both NUMALink4 and InfiniBand are capable of very good performance. While the HPC Challenge benchmarks showed some potential performance problems with InfiniBand, those results were not seen with either the NPBs or the applications we tested. Furthermore, we note that because of the limitations of the InfiniBand hardware, fully employing more than three Altix nodes in a computation requires that a multilevel parallel programming paradigm be used. It is particular encouraging that application performance using InfiniBand on our maximum configuration was not trailing off.

For the final version of this paper, we will perform more experiments to compare the relative performance of NUMAlink4 versus InfiniBand; for example, we want to complete the multinode version of INS3D to use it for testing. In addition, we would like to try and pin down the causes of some of the anomalies we have seen with InfiniBand performance.

In future work, we will explore the causes of scaling problems that we observed with OpenMP. We will also experiment with the SHMEM library, including porting INS3D to use it.

Acknowledgements

We would like to thank Bron Nelson, Davin Chan, Bob Ciotti, and Bill Thigpen for their assistance in using Columbia. Rob Van der Wijngaart played a critical role in developing the multi-zone NPBs.

References

[1] D. Bailey, J. Barton, T. Lasinski, and H. S. (Eds.). The NAS Parallel Benchmarks. Technical Report NAS-91-002, NASA Ames Research Center, Moffett Field, CA, 1991.

[2] D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[3] P. G. Buning, D. C. Jespersen, T. H. Pulliam, W. M. Chan, J. P. S. amd S. E. Krist, and K. J. Renze. Overflow user's manual, version 1.8g. Technical report, NASA Langley Research Center, Hampton, VA, 1999.

[4] M. J. Djomehri and R. Biswas. Performance analysis of a hybrid overset multi-block application on multiple architectures. In *Proc. High Performance Computing - HiPC 2003, 10th International Conference*, Hyderabad, India, December 2003.

[5] M. J. Djomehri, R. Biswas, and N. Lopez-Benitez. Load balancing strategies for multi-block overset grid applications. In *Proc. 18th International Conference on Computers and Their Applications*, pages 373–378, Honolulu, HI, March 2003.

[6] Effective Bandwidth Benchmark. http://www.hlrs.de/organization/par/services/models/mpi/b_eff/.

[7] HPC Challenge Benchmarks. http://icl.cs.utk.edu/hpcc/.

[8] InfiniBand Specifications. http://www.infinibandta.org/specs.

[9] H. Jin and R. Van der Wijngaart. Performance characteristics of the multi-zone NAS Parallel Benchmarks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, NM, April 2004.

[10] C. Kiris, D. Kwak, and W. Chan. Parallel unsteady turbopump simulations for liquid rocket engines. In *Supercomputing 2000*, November 2000.

[11] C. Kiris, D. Kwak, and S. Rogers. Incompressible Navier-Stokes solvers in primitive variables and their applications to steady and unsteady flow simulations. In M. Hafez, editor, *Numerical Simulations of Incompressible Flows*. World Scientific, 2003.

[12] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. Panda. Performance comparison of MPI implementations over InifinBand, Myrinet, and Quadrics. In *Proceedings of SC'03*, Phoenix, AZ, November 2003.

[13] R. Meakin and A. M. Wissink. Unsteady aerodynamic simulation of static and moving bodies using scalable computers. In *Proc. 14th AIAA Computational Fluid Dynamics Conference*, Paper number 99-3302, Norfolk, VA, 1999.

[14] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB.

[15] D. C. Rapport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1995.

[16] R. Strawn and M. Djomehri. Computational modeling of hovering rotor and wake aerodynamics. *Journal of Aircraft*, 39(5):786–793, 2002.

[17] J. R. Taft. Achieving 60 gflop/s on the production cfd code overflow-mlp. *Parallel Computing*, 27(4):521–536, 2001.

[18] Top500 Supercomputer Sites. http://www.top500.org.

[19] Voltaire ISR 9288 InfiniBand switch router. http://www.voltaire.com/documents/9288dsweb.pdf.